Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Màster en Enginyeria Informàtica

Herramienta de visualización de datos para aplicaciones de IoT

Autor:
Osmar Rafael Rodríguez Gutiérrez

Director:
Luís Domingo Velasco Esteban

Junio 18, 2019

Resumen

El Internet de las Cosas o mejor conocido como IoT por sus siglas en inglés, *Internet of Things*, tiene como premisa "conectar lo desconectado". Un ecosistema de IoT consiste en dispositivos inteligentes habilitados para la web que utilizan procesadores integrados, sensores y hardware de comunicación para recopilar, enviar y actuar sobre los datos que adquieren de sus entornos. Los dispositivos de IoT comparten los datos que recopilan los sensores al conectarse a una puerta de enlace de IoT u otro dispositivo donde los datos se pueden enviar a la nube para ser analizarlos o también pueden analizarse localmente.

Existen numerosas aplicaciones de Internet de las cosas en el mundo real, que van desde la IoT del consumidor hasta la IoT empresarial. Los dispositivos IoT producen muchos tipos de información, incluyendo telemetría, metadatos, estado, comandos y respuestas.

La monitorización en este contexto se refiere al uso de un conjunto de herramientas y procesos que ayudan a detectar, depurar y resolver los problemas que ocurren en los sistemas mientras estos están funcionando.

El objetivo de este proyecto es desarrollar una herramienta en la que al recolectar datos de sensores desplegados en un área, estos sean almacenados en una base de datos de series temporales (ej., Prometheus). La recolección de los datos estará basada en protocolos específicos, como MQTT sobre LoRa. Luego, se representarán los datos en cuadros de mando que permitirán hacer análisis de los mismos.

Agradecimientos

Este trabajo no habría sido posible sin la guía y contribución de mi tutor, Dr. Luis Velasco, quién proporcionó su orientación y apoyo continúo durante todo el desarrollo de este trabajo.

A mis padres, a mi hermana y a toda mi familia, gracias por el amor y apoyo incondicional que me han brindado durante esta etapa académica.

A mis amigos, por convertirse en mi familia durante toda esta etapa y estar conmigo tanto en los buenos como en los malos momentos.

A Dios, por darme esta increíble oportunidad.

Índice

Índice de figuras

1 Introducción

El Internet de las Cosas, por sus siglas en inglés IoT (*Internet of Things*), es un conjunto de tecnologías y casos prácticos que no tiene una definición única. Un punto de vista válido plantea IoT como el uso de dispositivos conectados a una red, integrados en el entorno físico, para mejorar algún proceso existente o para permitir que se desarrollen posibilidades nuevas que antes no eran posibles.

Los sistemas de monitoreo son responsables de controlar la tecnología utilizada por una empresa o institución (hardware, redes y comunicaciones, sistemas operativos o aplicaciones, entre otros) para analizar su funcionamiento y rendimiento, además de detectar y alertar sobre posibles errores. Un buen sistema de monitoreo es capaz de monitorear dispositivos, infraestructuras, aplicaciones, servicios e incluso procesos de negocios.

En este proyecto utilizamos Docker como herramienta para crear, implementar y ejecutar los diferentes elementos que componen la arquitectura del sistema de monitoreo utilizando contenedores. Éste está compuesto por una red de comunicación LoRa, un bróker MQTT Eclipse Mosquitto, el conjunto de herramientas de monitoreo y alertas de sistemas Prometheus, un *gateway* de código abierto llamado MQTTGateway que permite la comunicación entre Mosquitto y Prometheus, y Grafana como suite de visualización y análisis de los datos obtenidos.

1.1 Objetivos

El IoT permite automatizar y controlar las tareas que se realizan a diario, evitando la intervención humana. La visualización de datos proveniente de aplicaciones IoT (sistemas de monitoreo) proporciona información en tiempo real que puede ser utilizada e influir en la toma de decisiones dentro de una empresa o institución.

El objetivo de este proyecto es desarrollar una herramienta que permita recolectar datos de sensores desplegados en un área y almacenarlos en una base de datos de series temporales (ej., Prometheus). La recolección de los datos estará basada en protocolos específicos, como MQTT sobre LoRa. Se diseñarán diferentes vistas de los datos que permitan tener control sobre el estado de los sensores.

Para lograr esto, los principales objetivos que proponemos en este trabajo son:

- Establecer la arquitectura del sistema y determinar los módulos necesarios para que pueda ser implementado.
- Definir las herramientas que permitirán el despliegue y ejecución de lo módulos del sistema.
- Diseñar un tablero que permita visualizar los datos provenientes de los sensores.

1.2 Motivación y trabajo relacionado

Son cada vez más los dispositivos inteligentes presentes en el mercado, y esta tendencia se irá profundizando cada vez más. Por tanto, es importante conocer como utilizar los datos captados por estos aparatos ya que permitirán simplificar tareas o ayudar en la toma de ciertas decisiones. Me interesé en este proyecto con la meta de mejorar mi conocimiento sobre este tipo de tecnología, desarrollando este proyecto y cumpliendo con los objetivos propuestos.

Este proyecto parte de esta necesidad de captar y visualizar la información obtenida por dispositivos inteligentes. El desarrollo de la red de comunicación LoRa es parte de otro proyecto y por tanto, esta será la fuente de entrada de datos del sistema de visualización de datos que se diseñará.

1.3 Estructura del proyecto

Este proyecto está organizado de la siguiente manera. En primer lugar, presentamos una descripción general sobre el Internet de las Cosas (IoT por si siglas en inglés) y de los sistemas de monitoreo. En la sección 3, presentamos diferentes tipos de soluciones para temas de monitoreo de datos, almacenamiento y visualización. La descripción de la arquitectura de la solución y las herramientas que se han utilizado para desarrollar el trabajo se exponen en la sección 4. En la sección 5, presentamos la implementación y escenarios de prueba, así como los resultados obtenidos. Finalmente, concluimos el trabajo en la sección 6.

2 Descripción general del Internet de las Cosas (IoT)

El Internet de las Cosas o mejor conocido como IoT por sus siglas en inglés, *Internet of Things*, tiene como premisa "conectar lo desconectado". Esto quiere decir que objetos que no están actualmente conectados a una red de computadoras, entiéndase, a Internet, se conectarán de manera que puedan comunicarse e interactuar con personas y otros objetos. IoT es una transición tecnológica en la que los dispositivos permitirán detectar y controlar el mundo físico al hacer que los objetos sean más inteligentes y conectarlos a través de una red inteligente.

El mundo de IoT es amplio y puede resultar un tanto complicado debido a la gran cantidad de componentes y protocolos que abarca. En lugar de ver IoT como un dominio de tecnología único, es bueno verlo como un paraguas de varios conceptos, protocolos y tecnologías, que a veces dependen en cierta medida de una industria en particular. Si bien la amplia gama de elementos de IoT está diseñada para crear numerosos beneficios en las áreas de productividad y automatización, al mismo tiempo presenta nuevos desafíos, como escalar la gran cantidad de dispositivos y cantidades de datos que deben procesarse.

La persona acreditada con la creación del término "Internet de las cosas" es Kevin Ashton. Mientras trabajaba para Procter & Gamble en 1999, Kevin utilizó esta frase para explicar una nueva idea relacionada con la vinculación de la cadena de suministro de la empresa a Internet.

IoT se ha desarrollado a partir de la convergencia de tecnologías inalámbricas, sistemas micro-electromecánicos (MEMS), micro-servicios e internet. Por ello, IoT es el resultado de la evolución de la comunicación máquina a máquina (M2M), *Machine-to-Machine*. Permite a una red de sensores de miles de dispositivos inteligentes conectar personas, sistemas y otras aplicaciones para recopilar y compartir datos.

2.1 Descripción general del Internet de las Cosas (IoT)

Un ecosistema de IoT consiste en dispositivos inteligentes habilitados para la web que utilizan procesadores integrados, sensores y hardware de comunicación para recopilar, enviar y actuar sobre los datos que adquieren de sus entornos. Los dispositivos de IoT comparten los datos que recopilan los sensores al conectarse a una puerta de enlace de IoT u otro dispositivo donde los datos se pueden enviar a la nube para ser analizados o también pueden analizarse localmente. A veces, estos dispositivos se comunican con otros dispositivos relacionados y actúan sobre la información que obtienen unos de otros. Los dispositivos realizan la mayor parte del trabajo sin intervención humana, aunque las personas pueden interactuar con ellos, por ejemplo, para configurarlos, darles instrucciones o acceder a los datos.

Los protocolos de conectividad, redes y comunicación utilizados con estos dispositivos habilitados para la web dependen en gran medida de las aplicaciones específicas de IoT implementadas.

2.2 Beneficios

El internet de las cosas ofrece una serie de beneficios a las organizaciones, permitiéndoles:

- Monitorear sus procesos comerciales generales.
- Disminuir costos, gracias a la utilización eficiente de los recursos.
- Aumento de la productividad, por la reducción de los esfuerzos humanos.
- Datos de alta calidad.
- Mejorar la experiencia del cliente, *marketing* en tiempo real.
- Integrar y adaptar modelos de negocio.
- Tomar mejores decisiones de negocios.

IoT alienta a las empresas a repensar las formas en que se acercan a sus negocios, industrias y mercados y les brinda las herramientas para mejorar sus estrategias comerciales.

2.3 Aplicaciones para consumidores y empresas

Existen numerosas aplicaciones de Internet de las cosas en el mundo real, que van desde la IoT del consumidor y la IoT empresarial hasta la IoT de manufactura e industrial (IIoT). Las aplicaciones de IoT abarcan numerosas áreas, incluyendo automotriz, telecomunicaciones, energía y entre otras.

En el segmento de consumidores, por ejemplo, los hogares inteligentes que están equipados con termostatos inteligentes y otros dispositivos inteligentes, ya sean de calefacción o iluminación, y pueden controlarse de forma remota a través de computadoras, teléfonos inteligentes u otros dispositivos móviles.

Los dispositivos portátiles también se usan para la seguridad pública, por ejemplo, para mejorar los tiempos de respuesta de los primeros rescatistas durante las emergencias al proporcionar rutas optimizadas a un lugar o al rastrear los signos vitales del personal, como por ejemplo los bomberos en sitios que amenazan la vida.

En salud, IoT ofrece beneficios como la capacidad de monitorear a los pacientes de forma más cercana y analizar los datos generados. Los hospitales a menudo usan sistemas de IoT para completar tareas como la gestión de inventarios, tanto para productos farmacéuticos como para instrumentos médicos.

Los edificios inteligentes pueden, por ejemplo, reducir costes en la energía utilizando sensores que detectan la cantidad de ocupantes que hay en una habitación, de esta manera ajustar automáticamente la temperatura. También, encender el aire acondicionado si los sensores detectan que la sala de conferencias está llena o apagar la calefacción si la oficina ha quedado vacía.

En la agricultura, los sistemas de agricultura inteligente basados en IoT pueden ayudar a monitorear la luz, temperatura, humedad de las plantas y del suelo de los campos de cultivo, así como controlar los sistemas de riego utilizando sensores conectados.

2.4 Desafíos de IoT

El Internet de las cosas conecta millones de dispositivos a Internet e implica el uso de miles de puntos de datos, todos los cuales necesitan ser protegidos.

Debido a que los dispositivos de IoT están estrechamente conectados, todo lo que tiene que hacer un pirata informático es explotar una vulnerabilidad para poder manipular todos los datos haciéndolos inutilizables. Cuando los fabricantes no actualizan sus dispositivos con regularidad, o en absoluto, dejan vulnerabilidades expuestas que pueden ser aprovechadas por los ciberdelincuentes.

Además, los dispositivos inteligentes a menudo solicitan a los usuarios que ingresen su información personal, incluidos nombres, edad, direcciones, números de teléfono e incluso cuentas de redes sociales, que son de gran valor para los piratas informáticos.

Sin embargo, los *hackers* no son la única amenaza para el internet de las cosas; la privacidad es otra preocupación importante para los usuarios de IoT. Por ejemplo, las compañías que fabrican y distribuyen dispositivos IoT a consumidores finales podrían utilizar estos para obtener y vender datos personales de los usuarios.

IoT y su gran número de sensores generan una número importante de datos que proporcionan información crítica y de mucho interés si pueden procesarse de manera eficiente. Sin embargo, el desafío es evaluar cantidades masivas de datos provenientes de diferentes fuentes en diversos formatos y que deben tratarse de manera eficaz.

Al igual que como cualquier otra tecnología naciente, existen varios protocolos y arquitecturas que están compitiendo por cuota de mercado y por convertirse en el estándar dentro de IoT. Algunos de estos protocolos y arquitecturas pertenecen a compañías privadas y otros son de código abierto. La definición de estándares de IoT está ayudando a minimizar este problema, pero todavía a menudo hay varios protocolos e implementaciones de diferentes proveedores disponibles en las redes IoT que puede dificultar la interconexión entre dispositivos.

3 Soluciones para temas de monitoreo de datos, almacenamiento y visualización

Los dispositivos IoT producen muchos tipos de información, incluyendo telemetría, metadatos, estado, comandos y respuestas.

La monitorización en este contexto se refiere al uso de un conjunto de herramientas y procesos que ayudan a detectar, depurar y resolver los problemas que ocurren en los sistemas mientras estos están funcionando.

En el caso del almacenamiento, existe una amplia gama de soluciones que permiten almacenar desde *blobs* (objetos binarios grandes) no estructurados de datos, como imágenes o flujos de video, hasta datos estructurados de dispositivos o transacciones.

Si bien existen herramientas que no están diseñadas específicamente para monitorear dispositivos IoT o procesos físicos, estas pueden aplicarse a partes que constituyen el monitoreo de IoT como la visualización de datos y señales.

3.1 Monitoreo de datos

El monitoreo está diseñado para proporcionar vistas o perspectivas sobre cómo una métrica cambia con el tiempo, no está diseñado para capturar todos los puntos de datos críticos o eventos. Esto contrasta con el registro del dispositivo (*device logging*), donde se listan los detalles de eventos específicos de un dispositivo individual.

El objetivo del monitoreo es ayudar a garantizar que una función o servicio se desempeñe según lo previsto.

Las mediciones realizadas en el punto de instrumentación dan como resultado el envío y registro de una métrica en el sistema de monitoreo centralizado. Las métricas pueden ser de nivel bajo (directo y sin procesar) o de nivel alto (resumen). Las métricas de nivel superior pueden calcularse a partir de las métricas de nivel inferior.

En general, unas métricas pueden considerarse adecuadas si tienen las siguientes características:

- Son accionables, es decir, informan a los que se encargan de operar o revisar el servicio cuando es necesario cambiar el comportamiento de alguna métrica.
- Son comparativas, es decir, se puede comparar el rendimiento de algo a lo largo del tiempo, o entre grupos de dispositivos cuyos miembros se encuentran en una ubicación diferente o tienen diferentes versiones de firmware o hardware.

- Son comprensibles y relevantes en un contexto operacional. Esto significa que, además de los valores en bruto como los totales, pueden proporcionar información como índices y tasas.
- Proporcionan información en la resolución correcta. Es posible elegir la frecuencia con la que se muestrea, la frecuencia con la que informa y la forma de promediar, agrupar y graficar sus métricas.

3.2 Almacenamiento

Los datos del mundo físico vienen en varias formas y tamaños. El estado de algunos dispositivos puede estar directamente conectado al hardware. Por ejemplo, cuando se verifica el estado de un pin digital, este se puede leer como ALTO o BAJO, según la lectura física del voltaje en el pin.

Otros estados de dispositivos pueden existir en la capa de aplicación. Por ejemplo, la grabación de audio puede tener una condición de estado de verdadero o falso, relacionado con si la aplicación está muestreando desde el micrófono o escribiendo en el disco.

Desde la perspectiva del software, el código de la aplicación que se ejecuta en el dispositivo mantiene la fuente de veracidad. A menudo es requerido que otro software lea el último estado conocido del dispositivo. Dado que los dispositivos IoT pueden pasar algo de tiempo en modo de suspensión de baja potencia y pueden existir en redes particularmente poco confiables, es habitual almacenar parte del estado de un dispositivo en la nube. De esa manera, los datos del estado pueden estar disponibles incluso cuando los dispositivos en sí están temporalmente fuera de línea.

Cuando se necesita que los datos del estado o telemetría estén disponibles para las aplicaciones móviles o web, es posible almacenar los datos procesados o sin procesar en bases de datos estructuradas pero sin esquemas. En las cuales se pueden representar los datos de los dispositivos IoT como objetos de dominio o nivel de aplicación.

3.3 Visualización

La materia prima del monitoreo es el almacenamiento en serie temporal o las mediciones de instrumentación, pero esta información sin procesar por sí sola no es muy útil directamente. Las visualizaciones ayudan a lograr una mejor comprensión del significado de los datos que se han obtenido.

El enfoque común es realizar algún tipo de agregación o consulta a través de los datos, y luego presentarlos visualmente. El objetivo de estas visualizaciones es ayudar y facilitar la toma de decisiones al poder interpretar los datos fácilmente.

Los gráficos deben ser fáciles de comprender, estos tienen que resaltar información clave y no ocultarla al tratar de hacer visualizaciones muy complejas. También, se pueden segmentar en diferentes tableros para hacer énfasis en el estado de ciertos datos y así poner especial atención, por ejemplo, en una rango temporal definido. Asimismo, es posible fraccionarlos, de manera tal, que se puedan presentar datos específicos relacionados al usuario está accediendo.

4 Arquitectura de la solución

Los desafíos y requisitos antes mencionados de los sistemas de IoT han impulsado una nueva disciplina de arquitectura de red. En los últimos años, han surgido estándares y marcos arquitectónicos para enfrentar el desafío de diseñar redes de IoT a gran escala.

Uno de los modelos más sencillos es la arquitectura IoT simplificada. Se caracteriza por destacar los bloques de construcción fundamentales que son comunes a la mayoría de los sistemas de IoT. Este marco de trabajo se presenta como dos pilas paralelas: la pila de gestión de datos de IoT y cómputo y la pila funcional central de IoT.

Fue escogida la pila funcional central de IoT como entorno de trabajo. La idea principal de este modelo es reducir la pila de la arquitectura IoT a tres capas, lo cual no implica que el sistema carezca del detalle necesario para desarrollar una estrategia sofisticada de IoT. Más bien, la intención es simplificar la arquitectura de IoT en sus bloques de construcción más básicos y luego usarla como base para comprender los principios clave de diseño e implementación que se aplican a los casos de uso específicos de cada industria.

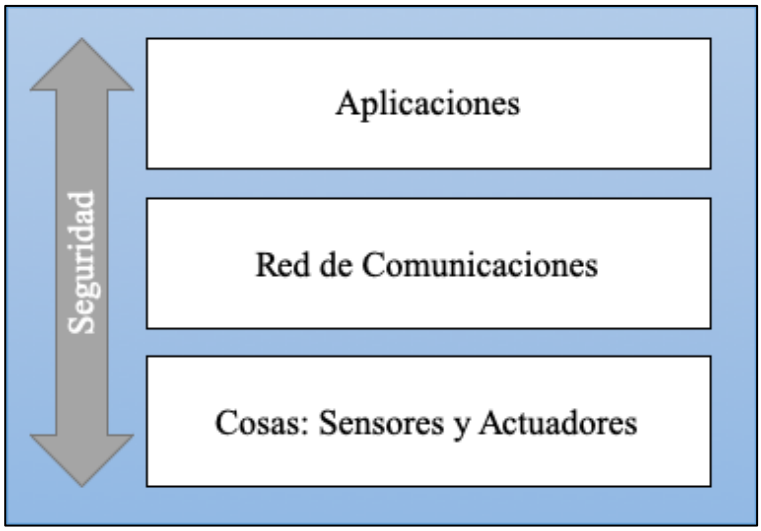

Figura 1. Pila funcional central de IoT

4.1 Plataforma

Docker es la herramienta escogida para realizar el despliegue del sistema. Docker es un proyecto de código abierto diseñado para facilitar la creación, implementación y ejecución de aplicaciones mediante el uso de contenedores. Los contenedores permiten a un desarrollador empaquetar una aplicación con todas las partes que necesita, como bibliotecas y otras dependencias, y distribuirla como un solo paquete. Al hacerlo, gracias al contenedor, el desarrollador puede estar seguro de que la aplicación se ejecutará en cualquier otra máquina,

independientemente de las configuraciones personalizadas que la máquina pueda tener y que puedan diferir de la máquina utilizada para escribir y probar el código.

Docker puede generar imágenes automáticamente al leer las instrucciones de un fichero Dockerfile. Este archivo es un documento de texto que contiene todos los comandos que un usuario puede llamar en la línea de comandos para ensamblar una imagen.

Además, Docker tiene una herramienta llamada Docker Compose que permite definir y ejecutar aplicaciones Docker de múltiples contenedores. Compose utiliza un archivo YAML para configurar los servicios de la aplicación y luego, con un solo comando, se puede crear e iniciar todos los servicios desde el fichero de configuración.

Esta herramienta fue utilizada para hacer el despliegue de la arquitectura desarrollada, la cual se enfoca específicamente en las capas de red de comunicaciones y aplicaciones.

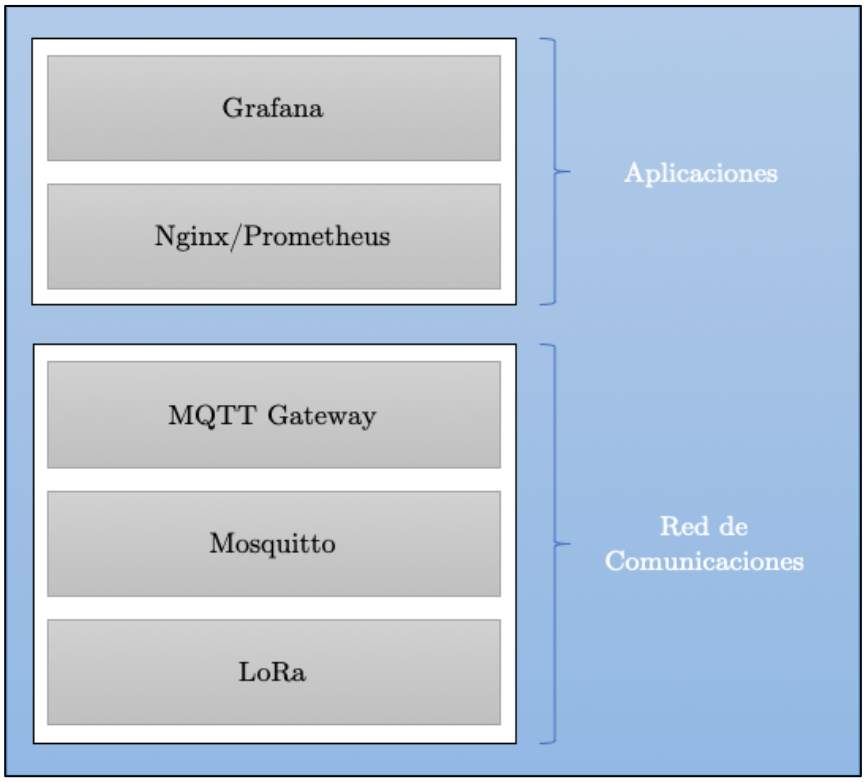

Figura 2. Arquitectura implementada

4.2 Cosas: Sensores y Actuadores

La mayoría de las redes de IoT comienzan desde el objeto que necesita estar conectado. Desde un punto de vista arquitectónico, la variedad de tipos de dispositivos inteligentes, formas y necesidades diferentes impulsan la variedad de protocolos y arquitecturas de IoT. Hay

innumerables formas de clasificar los objetos inteligentes. Una clasificación arquitectónica podría ser:

- Alimentado por batería o conectado a la alimentación: esta clasificación se basa en si el objeto lleva su propio suministro de energía o si recibe alimentación continua de una fuente de alimentación externa.
- Móvil o estático: esta clasificación se basa en si el dispositivo debe moverse o permanecer siempre en el mismo lugar. Un sensor puede ser móvil, por ejemplo, un sensor de viscosidad en una planta química o bien porque está conectado a un objeto en movimiento, por ejemplo, un sensor de ubicación en mercancías en movimiento. La frecuencia del movimiento también puede variar, de forma ocasional a permanente y el rango de movilidad desde unos pocos centímetros hasta kilómetros de distancia.
- Frecuencia de informe baja o alta: esta clasificación se basa en la frecuencia con la que el objeto debe informar los parámetros monitoreados. Las frecuencias más altas generan un mayor consumo de energía, lo que puede crear restricciones en la posible fuente de energía y el rango de transmisión.
- Datos simples o ricos: esta clasificación se basa en la cantidad de datos intercambiados en cada ciclo. Los datos más ricos generalmente generan un mayor consumo de energía. Esta clasificación a menudo se combina con la anterior para determinar el rendimiento de datos del objeto (rendimiento bajo a rendimiento alto).
- Rango de reporte: esta clasificación se basa en la distancia a la que se encuentra el sensor de la puerta de enlace.
- Densidad de objetos por celda: esta clasificación se basa en el número de objetos inteligentes en un área determinada, conectada a la misma puerta de enlace.

4.3 Red de comunicaciones

La segunda capa de la pila permite el almacenamiento local, el procesamiento de datos y la conexión a Internet. Los microcontroladores y la conectividad a Internet comparten la información capturada por los sensores dentro de los dispositivos de IoT y actúan en base a esa información para cambiar el entorno.

En esta entra tenemos la red de comunicación LoRa, MQTT-Mosquitto y el MQTT Gateway.

4.2.1 LoRa

LoRaWAN es un protocolo inalámbrico de largo alcance y baja potencia que está diseñado para su uso en la construcción de redes de IoT. Los dispositivos IoT ("nodos") envían pequeños paquetes de datos a número cualquiera de "puertas de enlace" que pueden estar en el rango

de varios kilómetros de un nodo a través del protocolo inalámbrico LoRaWAN. Luego, las puertas de enlace utilizan comunicaciones más tradicionales, como las conexiones de Internet por cable, para reenviar los mensajes a un servidor de red que valida los paquetes y reenvía la carga útil de la aplicación a un servidor de aplicaciones.

La naturaleza de la red LoRa permite que los dispositivos IoT funcionen durante años con baterías pequeñas, ocasionalmente enviando pequeños paquetes de datos, esperando un corto tiempo para recibir mensajes de respuesta y luego cerrando la conexión hasta que sea necesario enviar más datos. Los dispositivos también se pueden configurar para que siempre escuchen los mensajes de sus aplicaciones, aunque esto obviamente requiere más energía y puede ser más apropiado para dispositivos que, por ejemplo, están enchufados a una toma de pared.

Un red LoRa está compuesta por varios elementos:

- Dispositivos LoRaWAN: Son los dispositivos de IoT que envían datos a la red LoRa a través de las puertas de enlace de LoRa.
- Puerta de enlace LoRa: Recibe los datos de los dispositivos y generalmente ejecutan una implementación del software de reenvío de paquetes, este software es responsable de la interfaz con el hardware LoRa en la puerta de enlace.
- Puente de puerta de enlace LoRa: Es el responsable de la comunicación con la puerta de enlace, "transforma" el protocolo UDP del reenviador de paquetes en mensajes sobre MQTT.
- Servidor LoRa: Es el responsable de administrar el estado de la red. Tiene conocimiento de los dispositivos activos en la red y es capaz de manejar solicitudes de combinación cuando los dispositivos desean unirse a la red.
- Servidor de Aplicaciones LoRa: Proporciona una interfaz web y API para la gestión de usuarios, organizaciones, aplicaciones, puertas de enlace y dispositivos.

4.2.2 MQTT-Mosquitto

El protocolo MQTT es un protocolo de conectividad máquina a máquina (M2M) e IoT. MQTT es un protocolo de mensajería ligero que funciona con un mecanismo de suscripción-publicación basado en un bróker. Se ejecuta sobre el protocolo de control de transmisión / protocolo de Internet (TCP/IP).

En el mecanismo de publicación-suscripción, un cliente que publica un mensaje es desacoplado de otro cliente o clientes que reciben el mensaje, además los clientes no saben de la existencia de los otros clientes. Un cliente puede publicar mensajes de un tipo específico y solo los clientes que estén interesados en ese tópico específico recibirán los mensajes publicados.

El sistema de suscripción-publicación requiere de un bróker, también conocido como servidor, al que todos los clientes establecer una conexión. El cliente que envía un mensaje a través del bróker se le conoce como el publicador. El bróker filtra los mensajes entrantes y los distribuye a los clientes interesados en los tópicos de los mensajes recibidos. Los clientes que se registran en el bróker como interesados en un tópico específico se conocen como suscriptores. Por lo tanto, tanto los publicadores como los suscriptores establecen una conexión con el bróker.

Eclipse Mosquitto es un bróker de mensajes de código abierto (con licencia EPL / EDL) que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT. Entre los componentes de Mosquitto se pueden resaltar tres de ellos como los más básicos, `mosquitto.conf`, `mosquitto_sub` y `mosquitto_pub`.

`mosquitto.conf` es el archivo de configuración para Mosquitto. Este fichero puede estar en cualquier lugar del sistema siempre y cuando Mosquitto pueda leerlo, de manera predeterminada no necesita un archivo de configuración y utilizará los valores predefinidos en su código fuente. Entre las configuraciones predeterminadas de Mosquitto se puede señalar que utiliza el puerto 1883 para establecer la comunicación.

`mosquitto_sub` es un simple cliente de MQTT que se suscribe a tópicos e imprime los mensajes que recibe. Además de suscribirse a temas, `mosquitto_sub` puede filtrar los mensajes recibidos para que no se impriman (opción `-T`) o cancelar la suscripción de los temas (opción `-U`).

`mosquitto_pub` se refiere a un simple cliente de MQTT que publica un solo mensaje sobre un tema específico y luego se desconecta del bróker.

4.2.3 MQTT Gateway

El MQTT Gateway es un proyecto de código abierto escrito en lenguaje Go que permite transmitir los mensajes que se publican en un bróker MQTT a Prometheus. Para ello, esta herramienta se suscribe al bróker como un cliente más a un tópico específico para luego, una vez recibido un mensaje, redirigirlo hacia Prometheus. De forma predeterminada utiliza el puerto 9337.

4.4 Aplicaciones

El poder del Internet de las cosas aparece en las aplicaciones que hacen uso de la información intercambiada con los objetos inteligentes.

4.3.1 Prometheus

Prometheus es un conjunto de herramientas de monitoreo y sistemas de alertas de código abierto creado originalmente por SoundCloud, aunque hoy en día se mantiene independiente de cualquier compañía.

Las principales características de Prometheus son:

- Gestionar un modelo de datos multidimensional con datos de series de tiempo identificados por nombre de métrica y pares clave/valor.
- Utiliza PromQL, un lenguaje de consulta flexible para aprovechar esta dimensionalidad.
- La colección de series de tiempo ocurre a través de un modelo de extracción por medio de HTTP.
- Múltiples modos de gráficos y soporte de cuadros de mando.

La configuración de Prometheus se realiza mediante un fichero YAML, por defecto Prometheus tiene un archivo de configuración de muestra llamado `prometheus.yml`. Hay tres bloques de configuración en el archivo de configuración predeterminado: `global`, `rule_files` y `scrape_configs`.

El bloque `global` controla la configuración global del servidor Prometheus. Tenemos dos opciones presentes. El primero, `scrape_interval`, controla la frecuencia con la que Prometheus hará un barrido de los *targets*. La opción `evaluation_interval` controla la frecuencia con la que Prometheus evaluará las reglas, ya que Prometheus utiliza estas reglas para crear nuevas series de tiempo y generar alertas.

El bloque `rule_files` especifica la ubicación de las reglas que se cargarán en el servidor Prometheus.

El último bloque, `scrape_configs`, controla qué recursos supervisa Prometheus. Dado que Prometheus también expone datos sobre sí mismo como un punto final HTTP, puede rastrear y monitorear su propia salud. En la configuración predeterminada, hay un solo trabajo, llamado Prometheus, que elimina los datos de la serie de tiempo expuestos por el servidor Prometheus. El trabajo contiene un único destino, configurado estáticamente, el *localhost* en el puerto 9090. Prometheus espera que las métricas estén disponibles en la ruta de `/metrics`.

4.3.2 Nginx

Nginx es un software de código abierto para servidores web, proxy inverso, almacenamiento en caché, balanceo de carga, transmisión de medios y más. Además de sus capacidades de servidor HTTP, Nginx también puede funcionar como un servidor proxy para correo electrónico (IMAP, POP3 y SMTP), un sistema de proxy invertido y un equilibrador de carga para servidores HTTP, TCP y UDP.

Debido a que Prometheus no admite directamente la autenticación básica (también conocida como "*basic auth*") para las conexiones con el navegador de Prometheus y la API HTTP. Para ello, se debe utilizar Prometheus junto con un servicio de proxy invertido que aplique la autenticación en la capa de proxy como por ejemplo Nginx.

Para ejecutar una instancia de Prometheus detrás de un servidor Nginx que se ejecuta en un *localhost* específico, todos las conexiones de Prometheus se deben redireccionar hacia el puerto habilitado para Nginx.

4.3.3 Grafana

Grafana es una suite de visualización y análisis de métricas de código abierto. Es comúnmente usado para visualizar datos de series de temporales y análisis de aplicaciones, pero también es utilizado en otros áreas, incluidos sensores industriales, domótica, clima y control de procesos.

Los paneles de control son la base de Grafana, los cuales están compuestos por paneles individuales organizados en varias filas. Esta herramienta facilita la construcción de las consultas a los datos y la personalización de las propiedades de visualización para poder crear tableros de acuerdo a las necesidades del proyecto. Cada panel puede interactuar con los datos de cualquier fuente de datos configurada. Actualmente se pueden importar datos de InfluxDB, Graphite, OpenTSDB, Prometheus y Cloudwatch.

5 Soluciones para temas de monitoreo de datos, almacenamiento y visualización

Después de describir los servicios que componen la arquitectura del sistema, en la siguientes subsecciones se describirá la configuración de los componentes de la herramienta de monitoreo y los resultados obtenidos.

Para desarrollar este trabajo se utilizó Docker como plataforma para realizar el despliegue. Docker se puede ejecutar en una amplia variedad de sistemas operativos UNIX, Windows y macOS. Esta herramienta ofrece una gran cantidad de contenedores e imágenes listos para ser utilizados, así como también permite crear contenedores personalizados.

Todos los ficheros utilizados se pueden encontrar en la sección de apéndices.

5.1 Manejo de contenedores

Para hacer el despliegue del proyecto se emplearon tres contenedores. En el primero, se ejecutan MQTT-Mosquitto y el MQTT Gateway; en el segundo, Prometheus y Nginx; y en el tercero se encuentra Grafana.

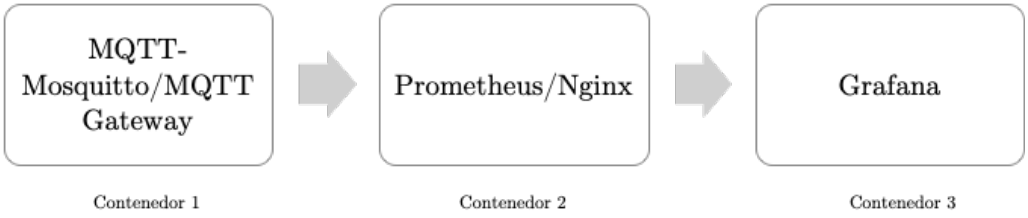

Figura 3. Estructura de contenedores

Se decidió desplegar todo el sistema utilizando Docker ya que permite simplificar su implementación e instalación, independientemente del sistema operativo que se utilice. También de esta manera evitar problemas de comunicación entre los diferentes componentes, ya que inicialmente hubo fallas de conexión entre los servicios que corrían en local y los servicios sobre Docker.

En algunos contenedores se ejecutan al menos dos servicios distintos, es por ello que se tuvo que hacer uso de un administrador de procesos como `supervisord`. Como se utilizaron contenedores personalizados fue necesario añadir los comandos de instalación de directamente en el fichero `dockerfile`. Adicionalmente se creó el fichero `supervisord.conf` donde se especifica la configuración del `supervisord`. Luego, a partir de los ficheros `dockerfile` se construyeron las imágenes con las que se crearon el primer y segundo contenedor.

Adicionalmente, se utilizó una red local personalizada para controlar la comunicación entre contenedores, y así habilitar la resolución automática de DNS de los nombres de los contenedores a las direcciones IP.

Todos los parámetros de configuración de cada contenedor y la red local se encuentran en el fichero `docker-compose.yml` el cual permite definir y ejecutar aplicaciones Docker de múltiples contenedores.

Por otra parte, se utilizó la herramienta Portainer para facilitar el manejo de los contenedores, redes, imágenes y volúmenes empleados por Docker.

5.1.1 Portainer

Portainer es una herramienta ligera y de código abierto que permite la gestión de contenedores sobre entornos Docker (Docker hosts y Docker Swarm). Portainer ofrece una interfaz gráfica para gestionar el host Docker desde cualquier navegador, tiene soporte para Raspberry Pi y se puede desplegar como un simple contenedor.

Portainer permite administrar todos los recursos de Docker (contenedores, imágenes, volúmenes, redes y más). Portainer no es una herramienta de monitorizado (a nivel de *host*), sino que está enfocada a la visualización básicamente del estado de los contenedores de uno (o varios) *endpoints* Docker (o Docker Swarm). Para acceder a él, simplemente se debe ir a `http://localhost:9000` en cualquier navegador.

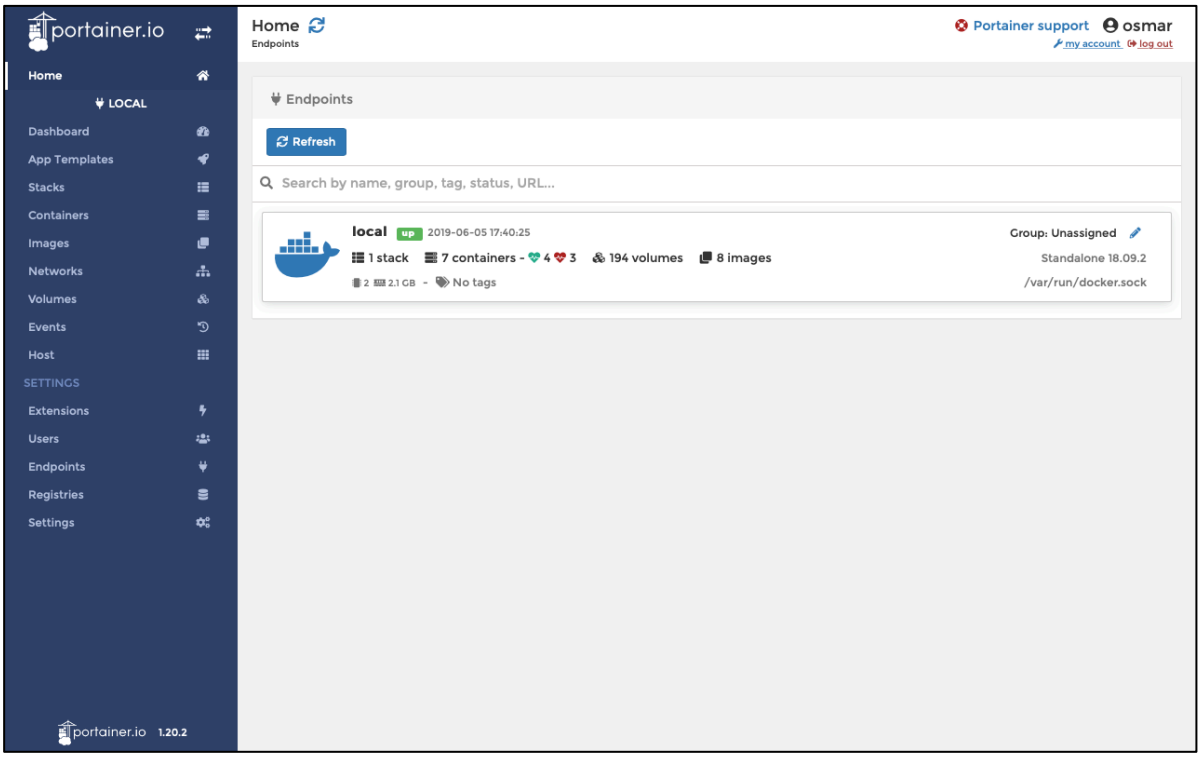

Figura 4. Pantalla principal de Portainer

En la Figura 4 se puede observar la pantalla principal de Portainer donde se puede escoger entre los diferentes *endpoints* de Docker disponibles.

Seguidamente, al seleccionar un *endpoint* específico se muestra la pantalla de la Figura 5 desde la cual se puede seleccionar entre los diferentes recursos disponibles.

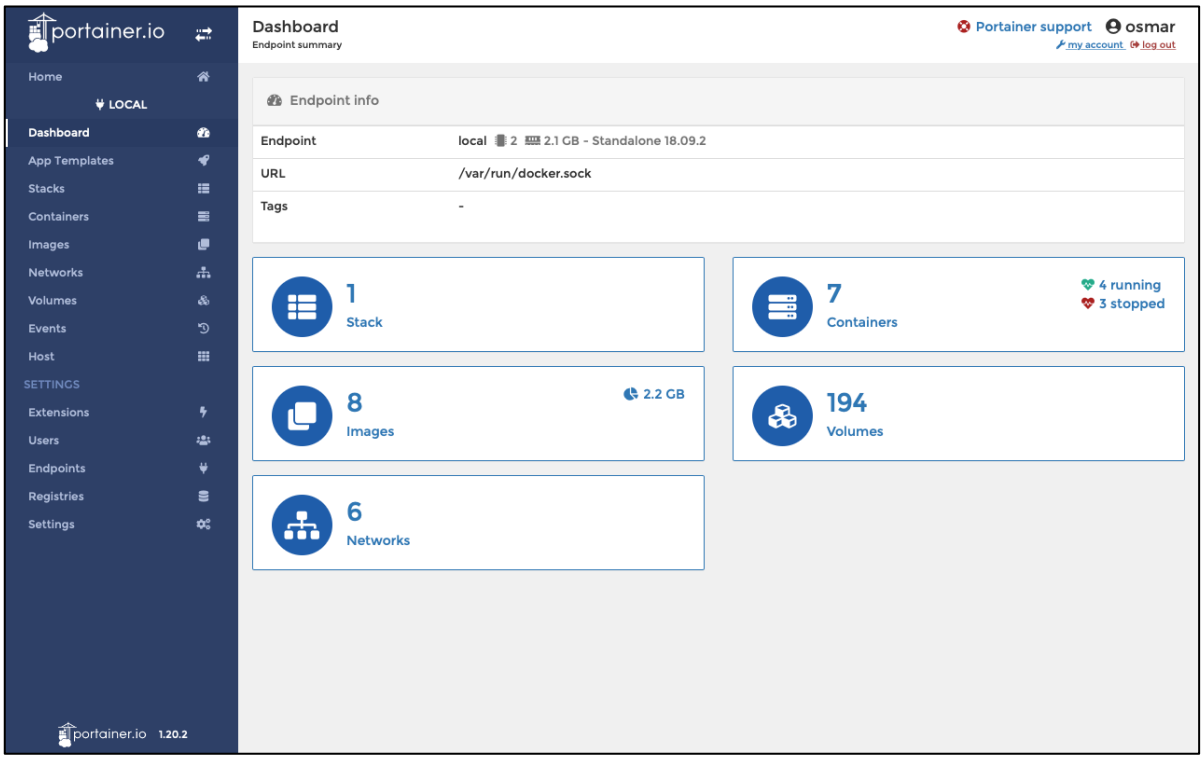

Figura 5. Resumen de los recursos disponibles

En las Figuras 6, 7, 8 y 9 se muestran las pantallas correspondientes al listado de contenedores, imágenes y redes disponibles en el *endpoint* de Docker seleccionado. En cada una de ellas se puede observar las diferentes tareas que Portainer permite realizar por medio de la interfaz gráfica sin tener que hacer uso de la línea de comandos. :

- Crear nuevos contenedores, imágenes, redes o volúmenes.
- Iniciar, detener, cerrar, reiniciar, pausar, reanudar y eliminar contenedores.
- Acceder al estado interno de cada uno de los contenedores.
- Construir, importar o exportar imágenes.
- Eliminar imágenes, redes o volúmenes.

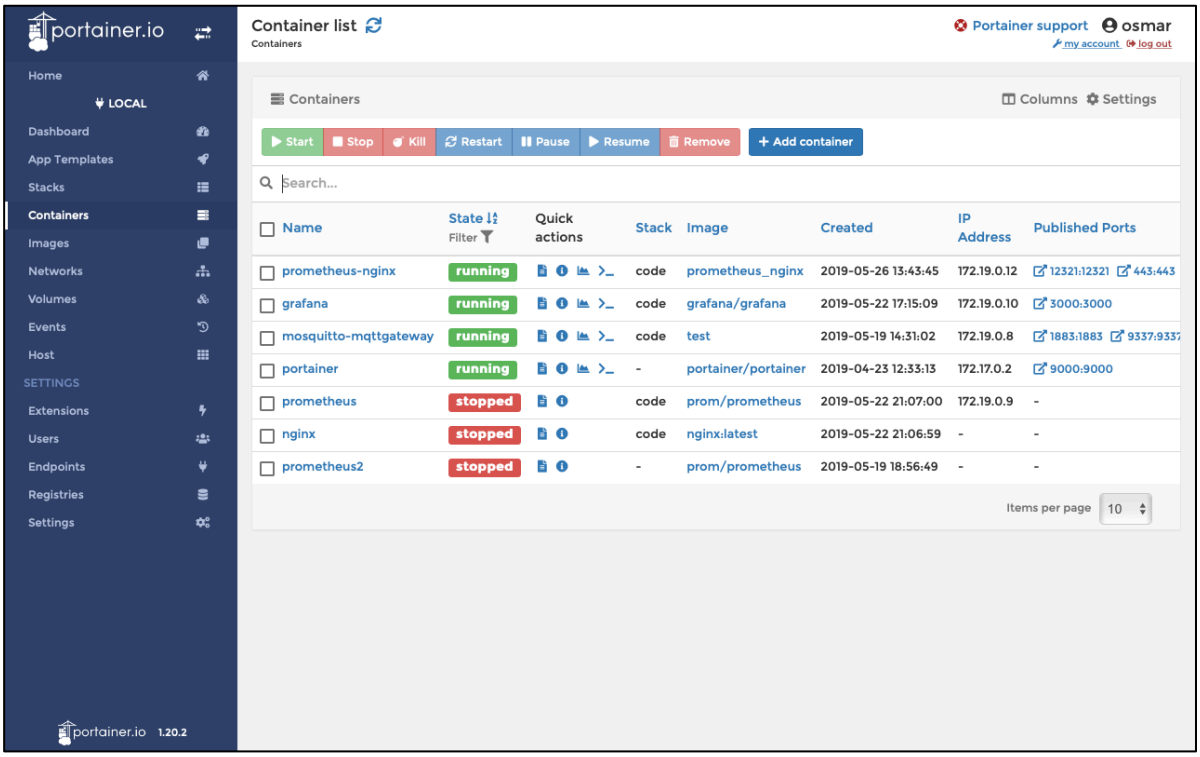

Figura 6. Listado de contenedores disponibles

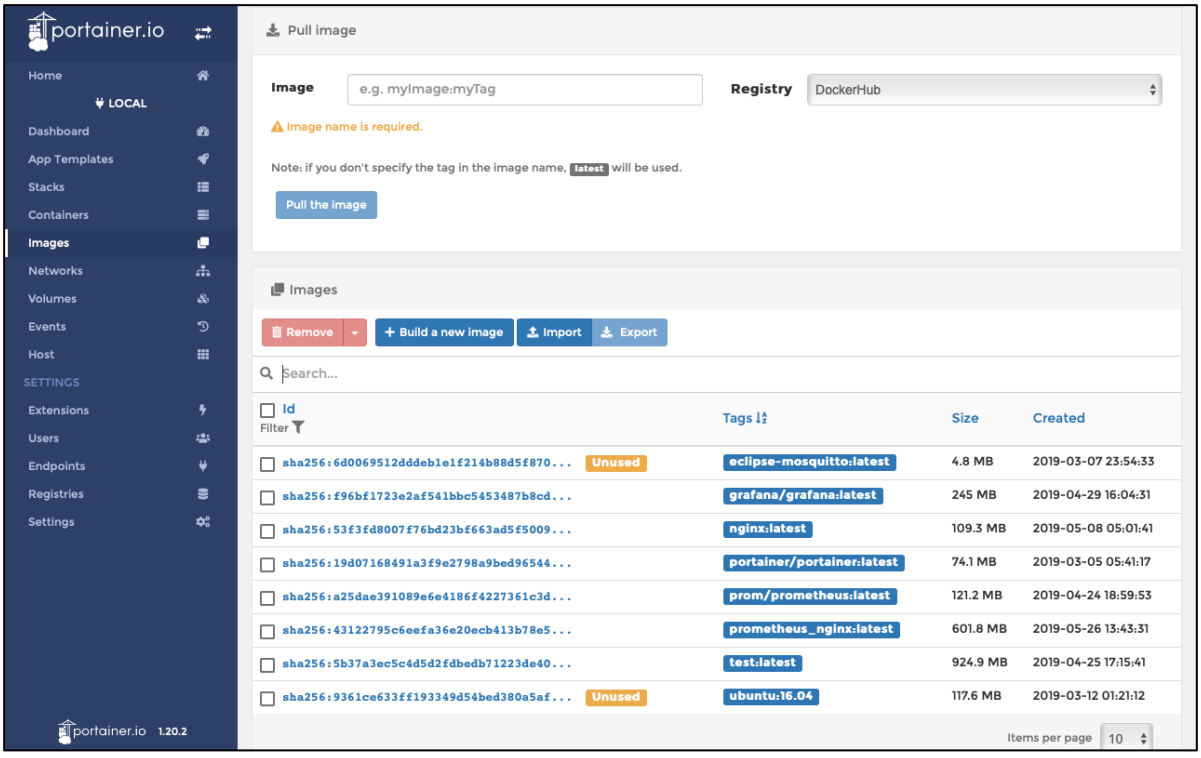

Figura 7. Listado de imágenes disponibles

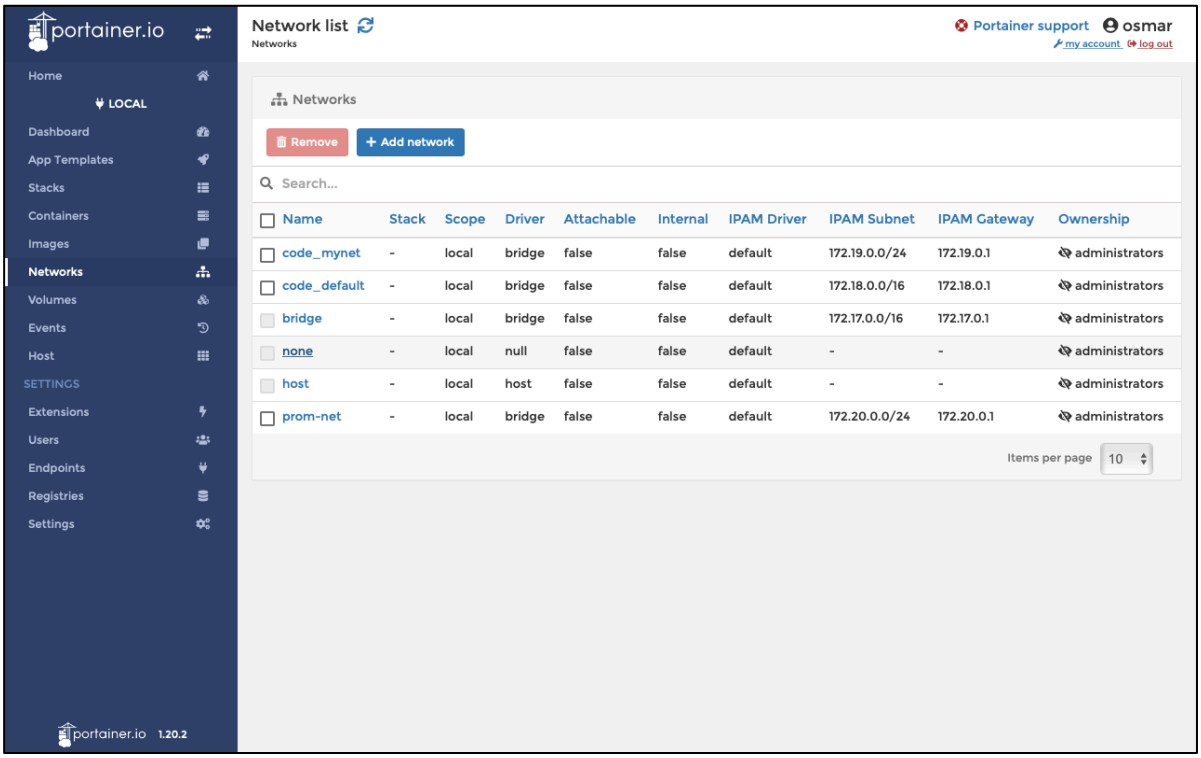

Figura 8. Listado de redes disponibles

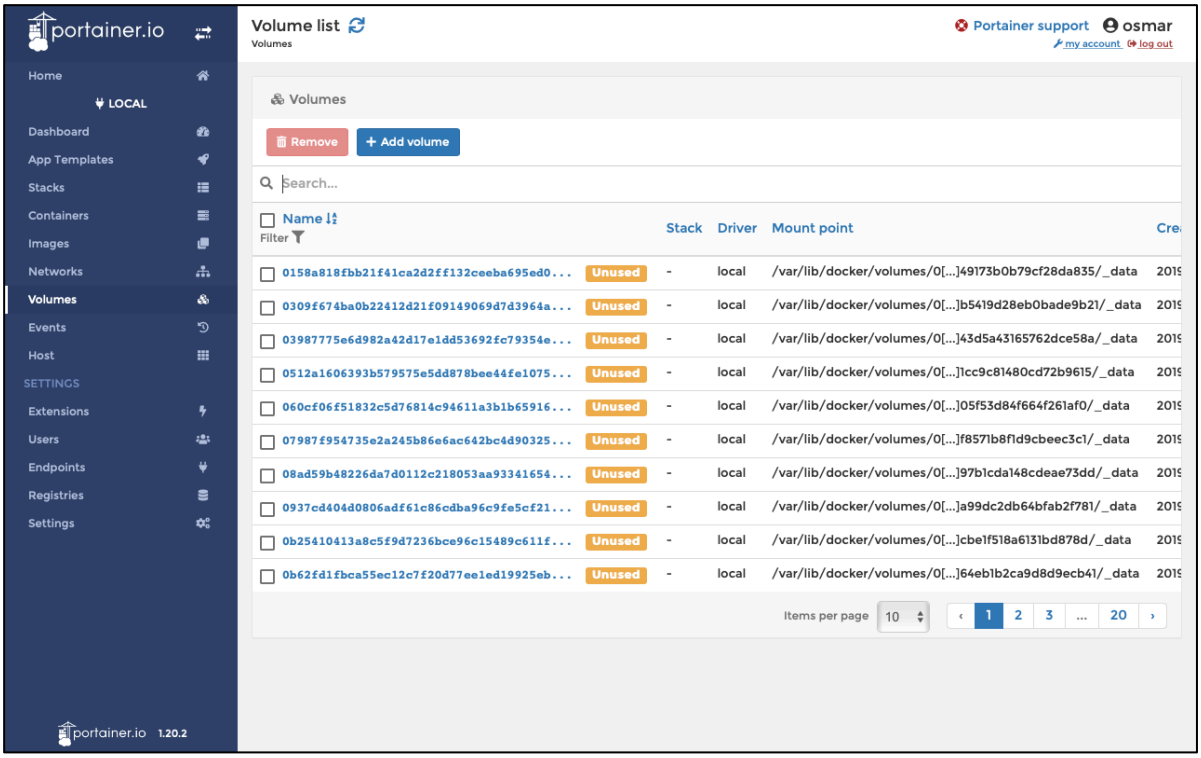

Figura 9. Listado de volúmenes disponibles.

5.1.2 MQTT-Mosquitto y MQTT Gateway

En este primer contenedor corren dos servicios, el bróker MQTT-Mosquitto y el MQTT *gateway* el cual permite enviar todos los mensajes publicados en el tópico "prometheus" hacia Prometheus.

Mosquitto está disponible en el puerto 1883 por el cual pueden subscribirse los dispositivos que generan los datos. Este bróker por si mismo no es capaz de almacenar la información que recibe, solamente permite distribuirla entre los diferentes clientes subscritos a él.

Por ello es necesario un servicio que permita almacenarla y así poder analizarla en tiempo real o posteriormente, de esta tarea se encarga el MQTT *Gateway*. El servicio está disponible en el puerto 9337 por el cual se pueden acceder a las métricas de la herramienta.

El MQTT Gateway depende de un bróker Mosquitto para poder funcionar, por tanto, si no hay ninguno disponible este automáticamente se suspende. Debido a esto, para evitar posibles problemas de conexión al iniciar un servicio primero que el otro, se decidió implementar ambos dentro de un mismo contenedor y de estar manera, por medio del `supervisord`, gestionar la puesta en marcha de cada uno de ellos de forma ordenada.

Utilizando Portainer es posible conocer el estado del contener de manera de saber si los servicios se están ejecutando correctamente.

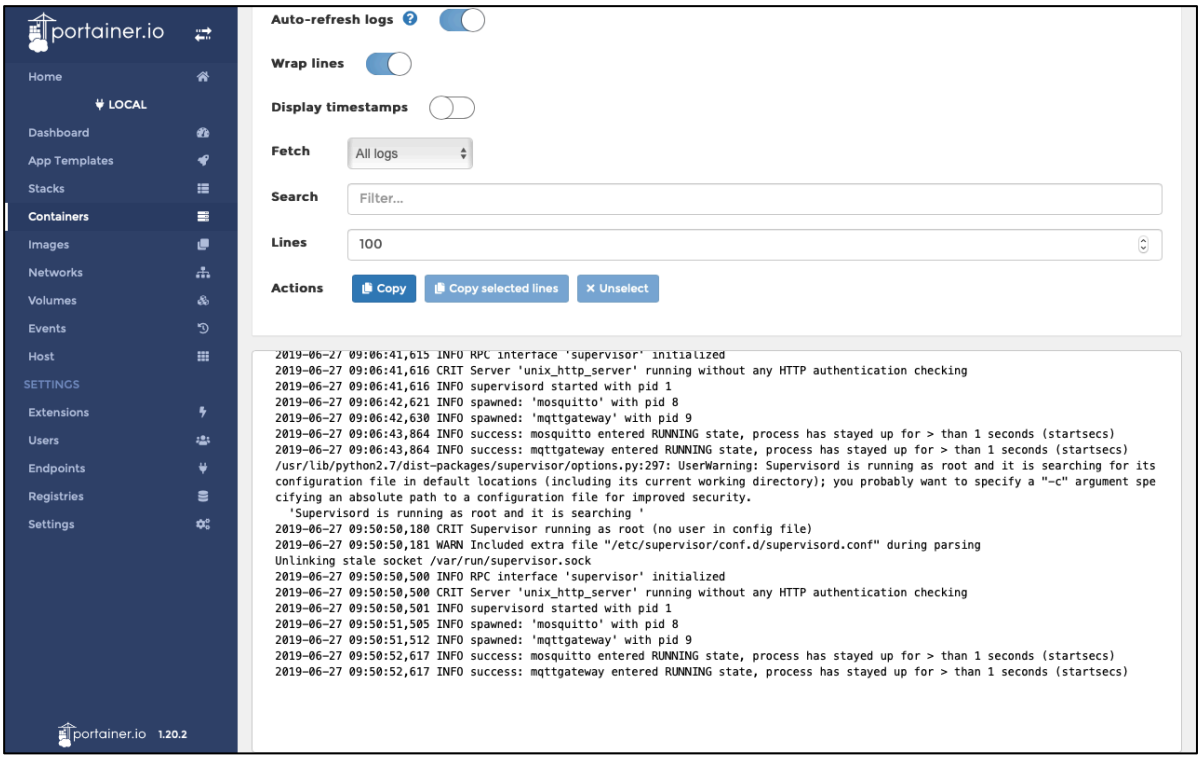

Figura 10. Estado del contenedor – MQTT-Mosquitto y MQTT Gateway

5.1.3 Prometheus y Nginx

En este segundo contenedor corren dos servicios, el conjunto de herramientas de monitoreo y alertas, Prometheus y Nginx el cual es utilizado como un sistema de proxy invertido que permite habilitar autenticación básica para acceder a Prometheus.

Primero que todo es necesario configurar Prometheus para que pueda conectarse con otros servicios. La configuración de Prometheus se realiza mediante un fichero YAML, por defecto Prometheus tiene un archivo de configuración de muestra llamado `prometheus.yml`. En el fichero se debe especificar el nombre del servicio al que Prometheus se conectará, la dirección IP del servicio y puerto en el que está disponible. En la Figura 11 se puede observar el estado de los servicios disponibles.

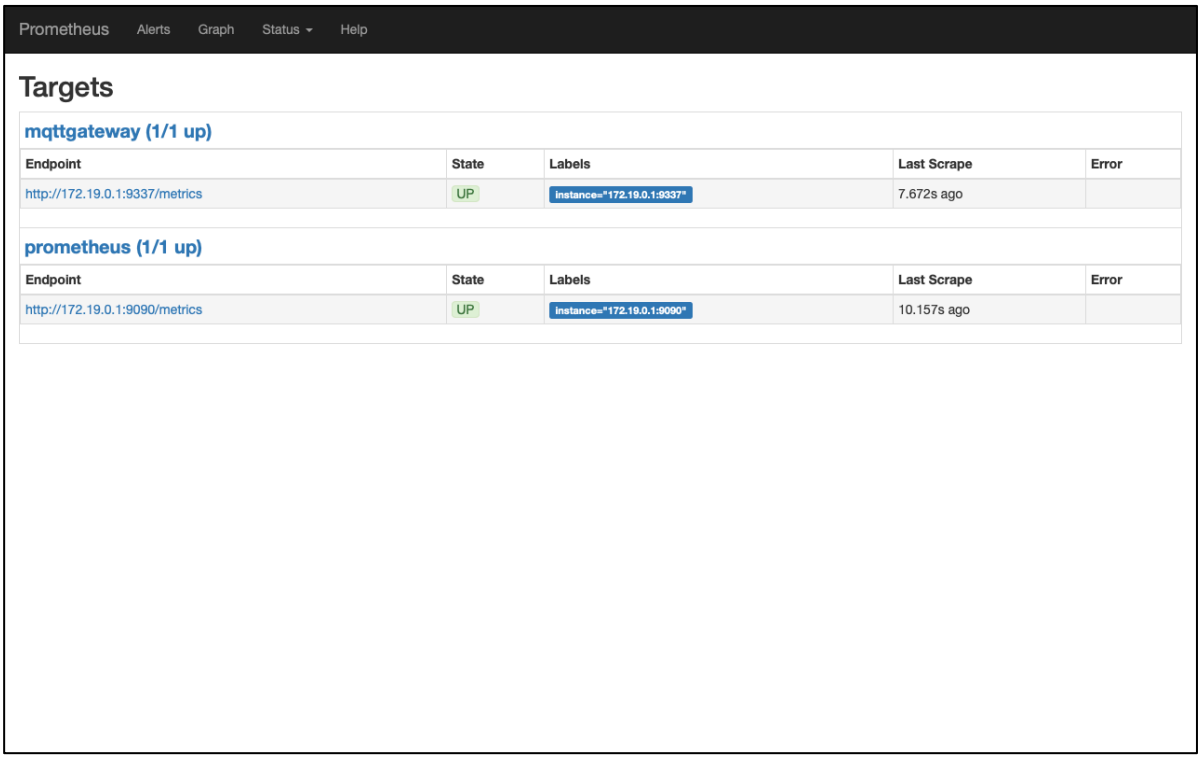

Figura 11. Estado de los servicios de Prometheus

Prometheus por si mismo no tiene una función de autenticación por usuario de contraseña, para ello, nos valemos de Nginx. Por este motivo, para poder solicitar un nombre de usuario y una contraseña a todos los usuarios que acceden a la instancia de Prometheus, es necesario un fichero con el nombre `.htpasswd`, en el cual se almacenan las credenciales creadas para cada de usuario al utilizar autenticación de acceso básica en un servidor HTTP Apache. Este fichero debe estar ubicado en los directorios de Nginx. Luego, es necesario modificar el archivo `nginx.conf`, donde se especifica la configuración de Nginx, para forzar la autenticación básica para todas las conexiones de la ruta `/prometheus`. En la Figura 12 se observa la ventada de

acceso a Prometheus como resultado de implementar la funcionalidad de autenticación básica a través de Nginx.

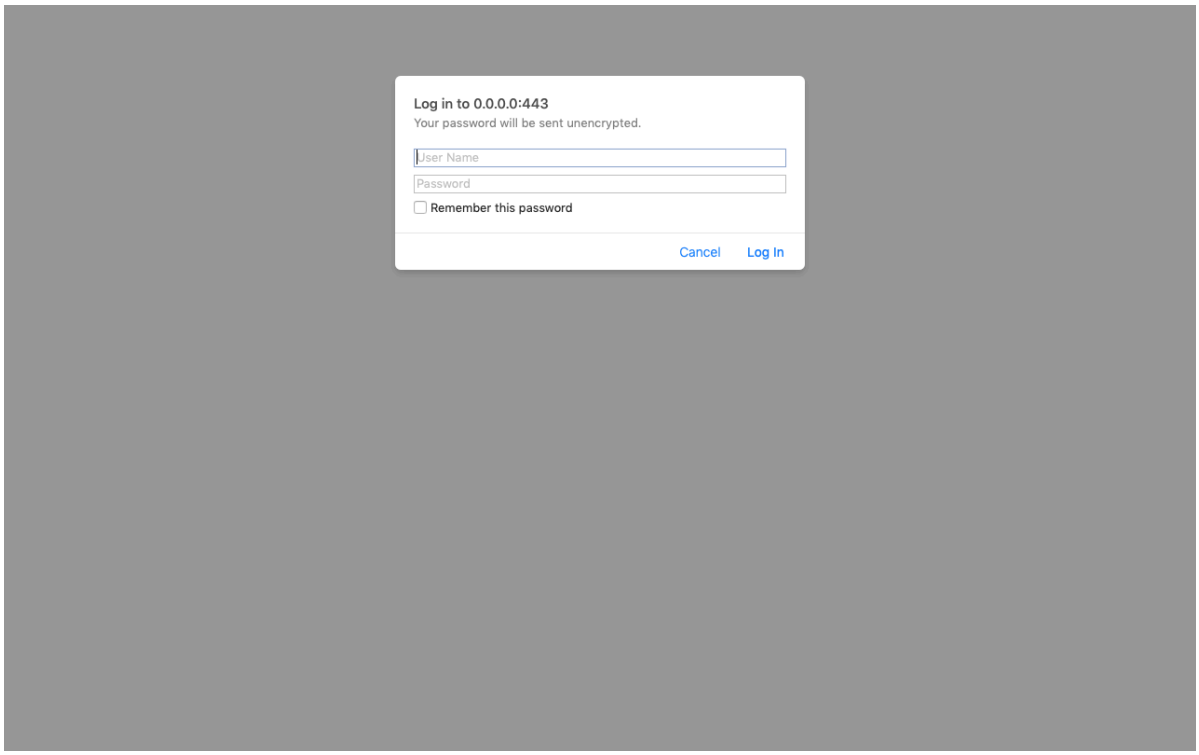

Figura 12. Acceder a Prometheus utilizando autenticación básica

Para poder correr Prometheus detrás del proxy Nginx es necesario indicar por medio del comando `--web.external-url` la dirección a la cual se debe redirigir el tráfico de la aplicación. Debido a esto, para evitar posibles problemas de direccionamiento se decidió implementar ambos servicios dentro de un mismo contenedor y de esta manera, por medio del `supervisord`, gestionar el flujo de datos de forma ordenada.

Utilizando Portainer es posible conocer el estado del contener de manera de saber si los servicios se están ejecutando correctamente, como se puede observar en la Figura 13.

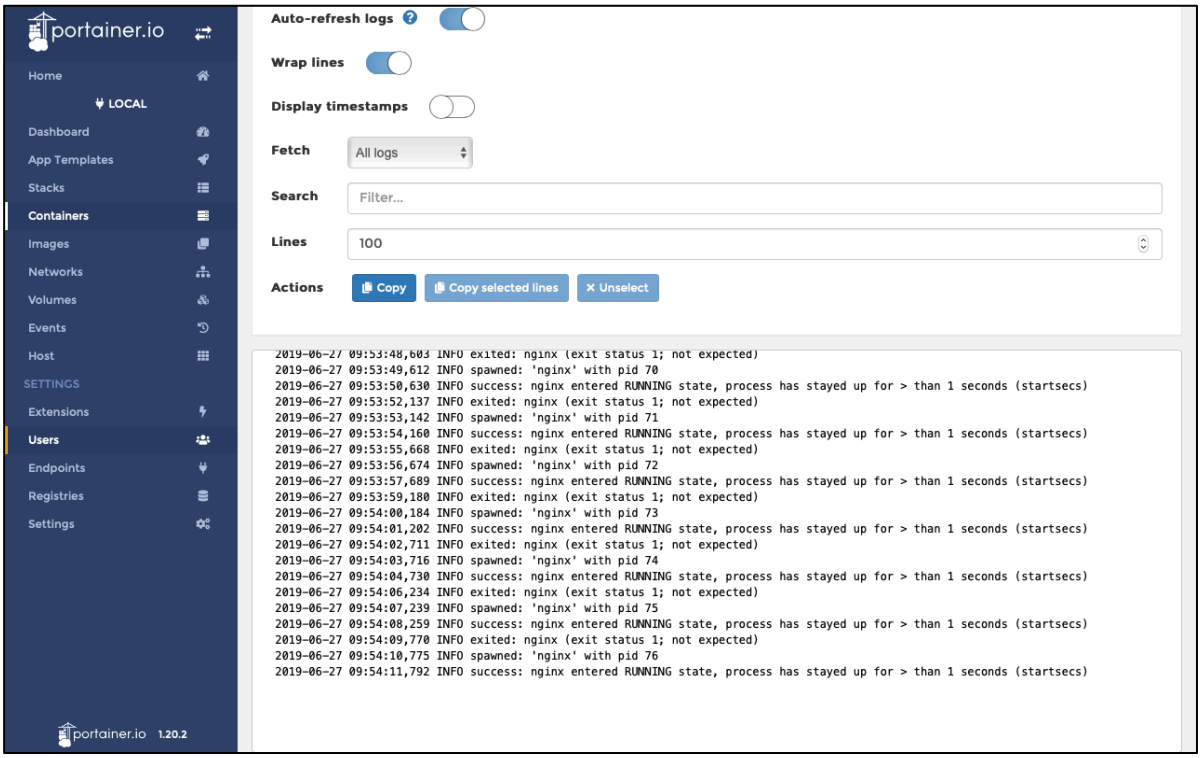

Figura 13. Estado del contenedor – Prometheus y Nginx

5.1.4 Grafana

En el tercer contenedor se ejecuta la suite de visualización y análisis de métricas, Grafana. El servicio posee su propio sistema de autenticación y se puede acceder a él a través del puerto 3000.

Con esta aplicación se realizaron los tableros para representar la data recibida por el bróker. En primer lugar se realizó la carga de la fuente de datos, en este caso Prometheus. Para ello se le debe indicar a Grafana la dirección (URL) en la que se encuentra Prometheus así como el tipo de autenticación utilizado y las credenciales para poder acceder, tal como se muestra en la Figura 14.

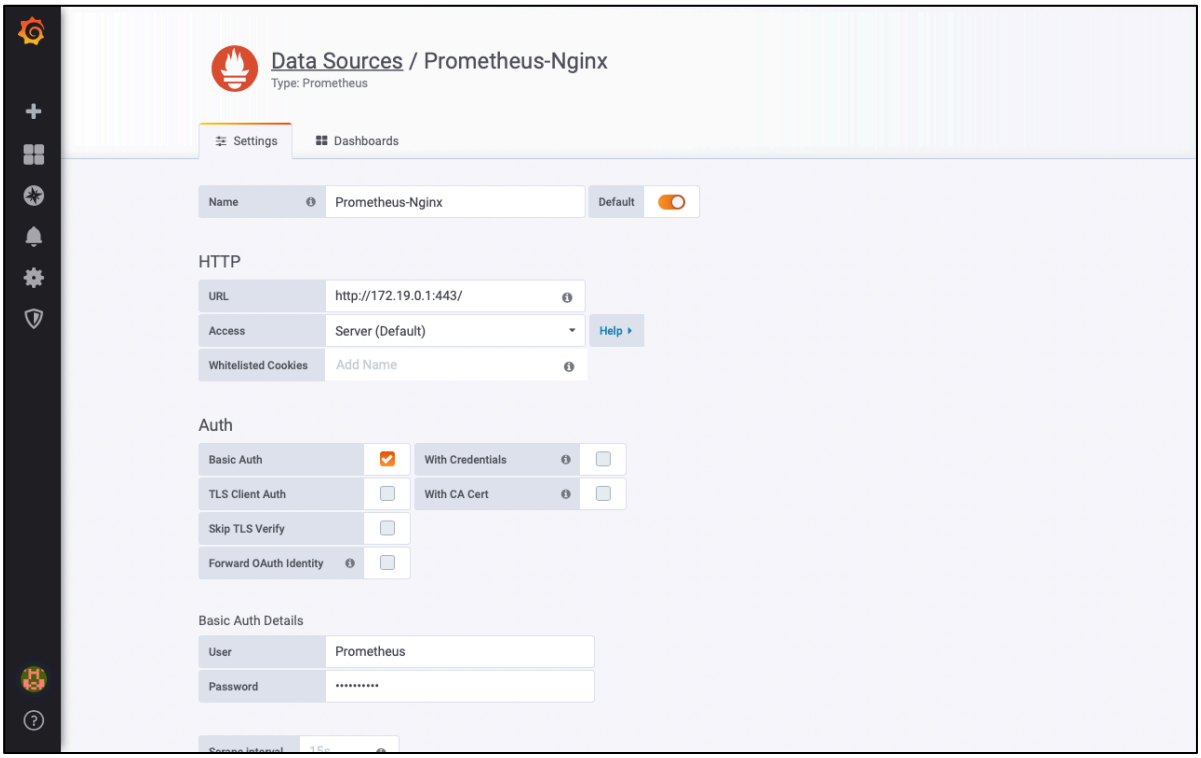

Figura 14. Prometheus como fuente de datos de Grafana

Luego de ello es posible crear los cuadros de mando que permitan visualizar la data que ha recibido Prometheus. Al crear un nuevo panel se puede escoger entre diferentes tipos de visualizaciones (gráfico, tabla, texto, mapa de calor, lista de alerta, cifra simple, *dashboard list*, *plugin list*). En las Figuras 15, 16, 17 y 18 se detallan las diferentes propiedades que se pueden modificar para cada panel.

En la pestaña de *Queries* se selecciona el dato que se quiere representar. En la pestaña *Visualization* se realiza la configuración de los ejes y se modifica la manera como se dibuja el modelo. En la pestaña *General* permite colocar un título y descripción al panel. En la pestaña de *Alert* se pueden crear alertas para algún panel específico cuando se sobrepasa o se baja de algún valor defino.

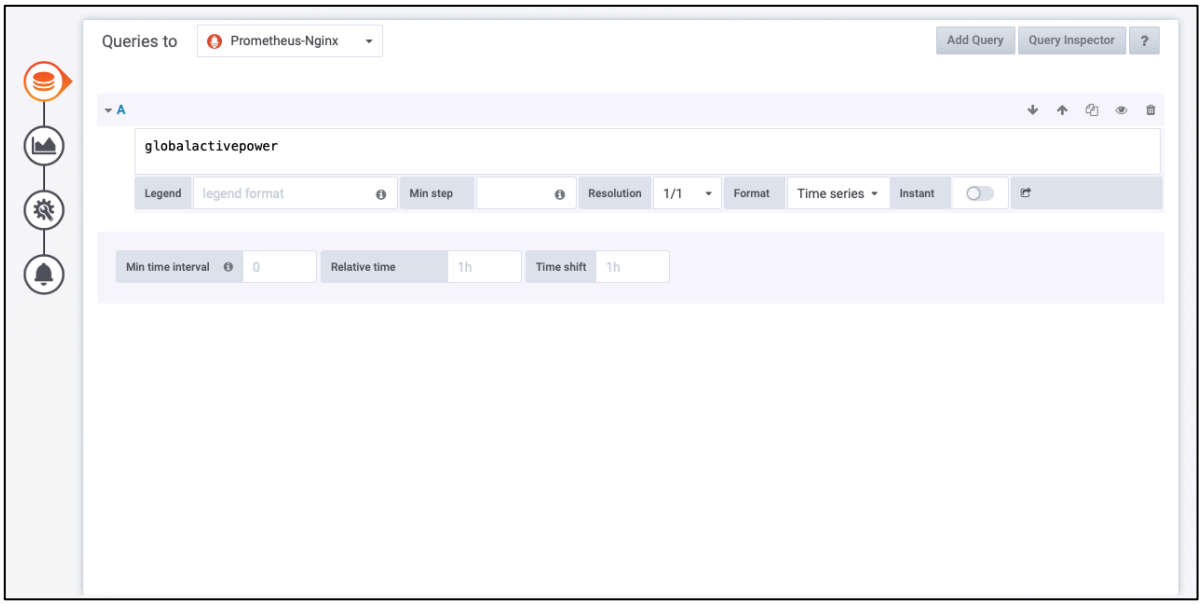

Figura 15. Pestaña panel – Queries

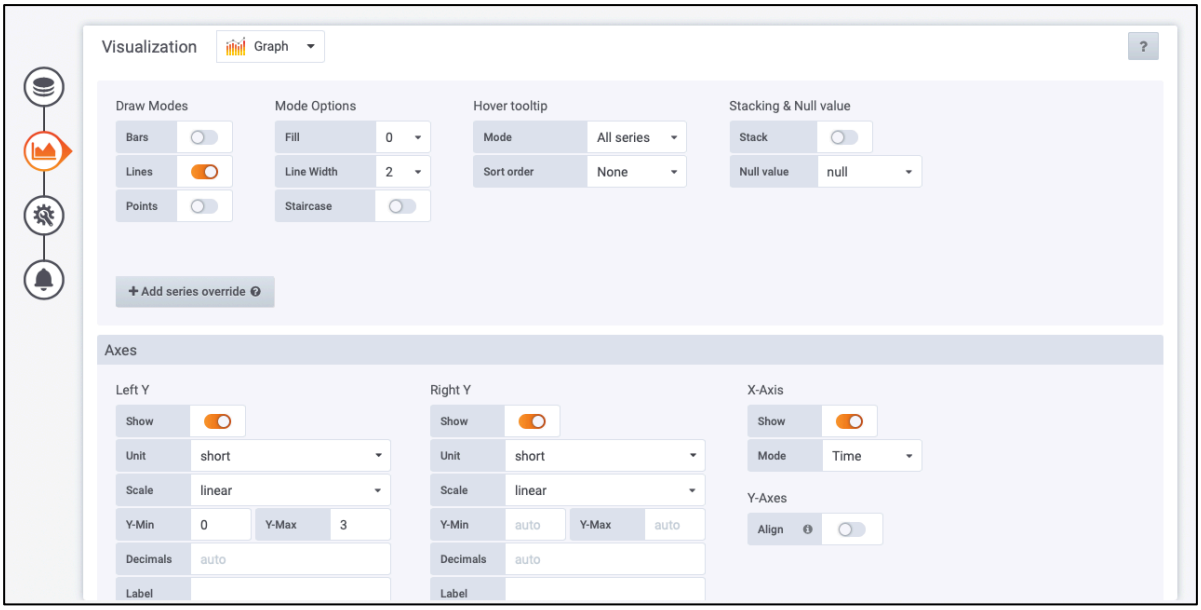

Figura 16. Pestaña panel – Visualization

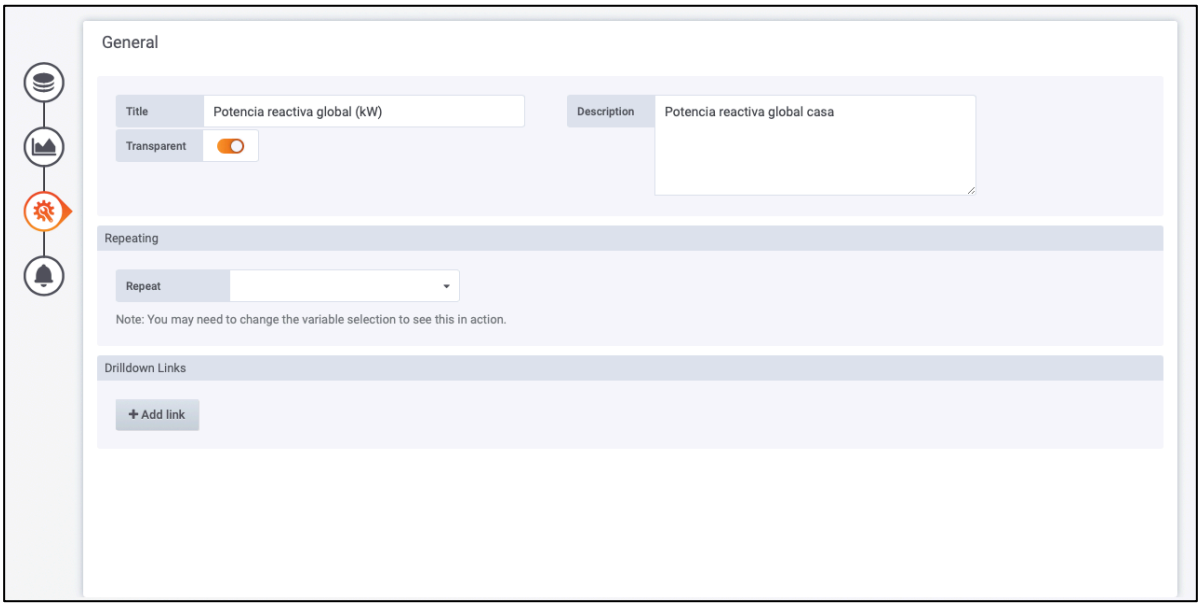

Figura 17. Pestaña panel – General

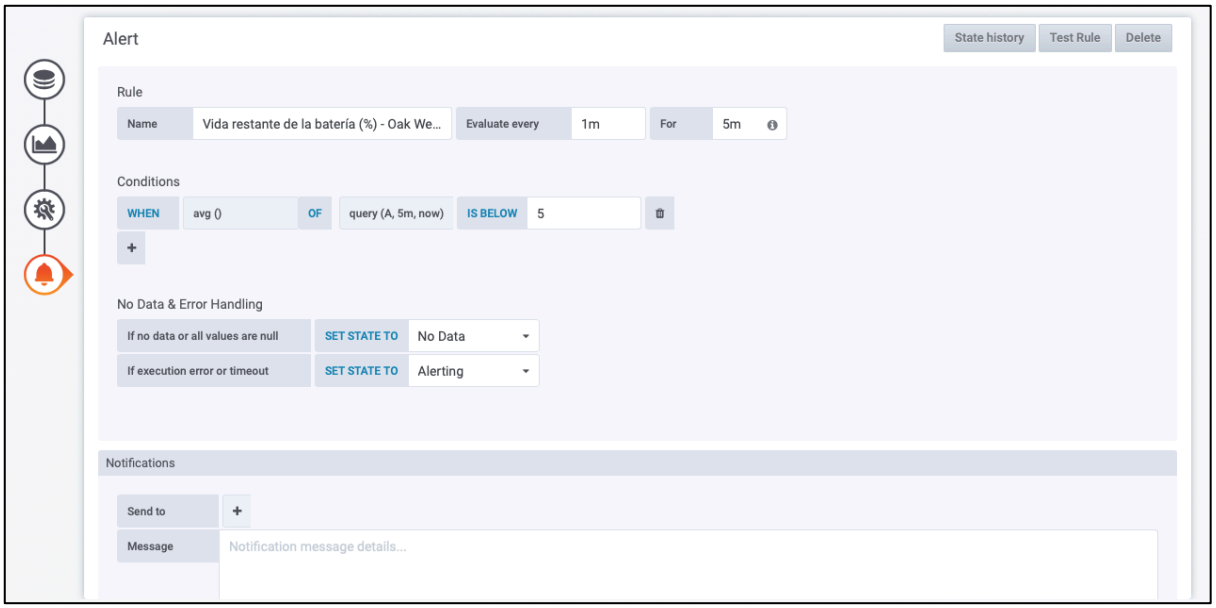

Figura 18. Pestaña panel – Alert

Utilizando Portainer es posible conocer el estado del contener de manera de saber si el servicio se está ejecutando correctamente, como se puede observar en la Figura 19.

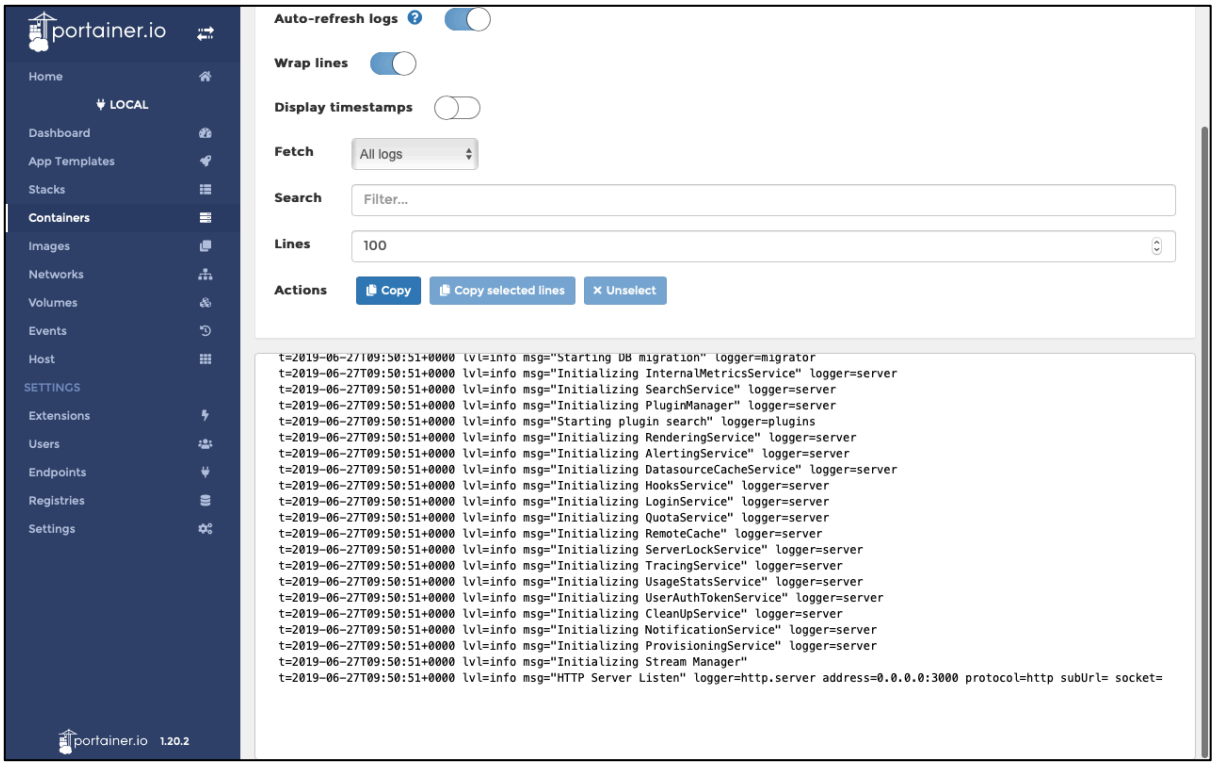

Figura 19. Estado del contenedor – Grafana

5.2 Cuadro de mando

Debido a problemas en la implementación de la red LoRa, ya que esta pertenecía a otro proyecto, se decidió utilizar como fuente de entrada de datos al sistema un programa en Python que lee datos de un fichero .csv. Para realizar las pruebas se utilizaron dos sub set de datos públicos obtenidos en el sitio web data.world. El primero es sobre el consumo de energía en un hogar entre enero y junio del año 2007, del cual se creó el cuadro de mando de la Figura 20.

Figura 20. Cuadro de mando - Consumo de energía en el hogar

En el cuadro de mando se pueden observar los gráficos correspondientes a la potencia activa global (kilovatios), la potencia reactiva global (kilovatios) y el voltaje (voltios), todos los valores han sido promediados en minutos. Del fichero de datos solo se han cogido las primeras 100 entradas a modo de muestra.

Este tipo de información es de gran utilidad para el análisis de sistemas eléctricos, ya que estas variables influyen en el cálculo del factor de potencia. Un factor de potencia fuera de rango conlleva a un menor rendimiento del suministro eléctrico, daña los equipos y maquinarias, y además, genera cargos adicionales en la facturación eléctrica.

También se realizó otra prueba con otro sub set datos públicos perteneciente al ayuntamiento de la ciudad de Chicago en Estados Unidos. La data proviene del *Chicago Park District* que mantiene sensores climáticos en las playas a lo largo del lago Michigan de Chicago, el período de los datos está entre mayo del 2015 y noviembre del 2017. En las Figuras 21 y 22 puede observarse el cuadro de mando creado.

Figura 21. Cuadro de mando - Parques de Chicago (parte 1)

Figura 22. Cuadro de mando - Parques de Chicago (parte 2)

En el cuadro de mando se pueden observar los gráficos correspondientes a las estaciones *Oak Street Weather Station* y *Foster Weather Station*. Para ambas estaciones se tomaron en cuenta los datos de temperatura del aire (Celsius), porcentaje de humedad relativa, total de precipitación recogida (milímetros), tipo de precipitación (0=sin precipitación; 60= precipitación líquida, aunque el hielo, granizo y aguanieve se transmiten como lluvia; 70 = precipitación sólida, 40 = precipitación no especificada), duración de la batería (voltaje de la batería, indica la vida útil restante de la batería). Es importante señalar para que la estación *Foster* la data sobre el total de precipitación recogida y tipo de precipitación no está disponible.

Esta información es de gran importancia para el ayuntamiento de Chicago para crear avisos o alertas a los ciudadanos sobre las condiciones climáticas de las playas, alertar si es seguro, restringir el acceso o generar recomendaciones específicas para su disfrute. Además, el indicador de la duración de la batería permite al *Chicago Park District* saber cuándo se deben reemplazar las baterías de los sensores para evitar tener interrupciones en el monitoreo. En general, este tipo de sistemas facilitan la toma decisiones a las autoridades de la ciudad y es un servicio que podría implementarse en cualquier ciudad del mundo.

6 Conclusiones

Este trabajo ha presentado una solución para realizar el sistema de monitoreo de datos de una red IoT. Como se ha mencionado anteriormente, no existe una solución genérica que pueda aplicarse a cualquier red. Los entornos de trabajos, *frameworks*, que existen actualmente deben tomarse como guías pero siempre deben ser adaptados a la particularidad de cada industria.

El objetivo principal era crear una herramienta de visualización de datos para aplicaciones IoT. La idea inicial era desplegar una parte del sistema de forma local y otra parte sobre Docker. Conforme iba avanzado el desarrollo de la herramienta, nos encontrábamos con problemas de comunicación entre los diferentes servicios que conforman el sistema. Principalmente esto se debía al hecho de que no todos los servicios se estaban implementando en una misma plataforma, y además, sin tomar el hecho que cada sistema operativo requiere una configuración específica para que puedan funcionar correctamente. Por este motivo, se decidió mover todo el despliegue hacia Docker para que fuese homogéneo y portable sin importar el sistema operativo.

El hecho de no disponer de una red LoRa para realizar la captura de datos de los sensores tuvo como consecuencia que tuviéramos que simular el envío de datos al sistema por medio de un programa escrito en Python para poder probar la herramienta de visualización.

A pesar de esto, mediante la simulación de los datos se pudieron crear los cuadros de mandos correspondientes a la data recibida. Este tipo de herramientas son de gran utilidad a los usuarios para poder tener una visión global de lo que ocurre en su red IoT y así poder realizar toma de decisiones más precisas.

Este tipo de herramientas en los próximos años serán cada vez más necesarias gracias a la progresiva puesta en funcionamiento de las redes de telefonía móvil 5G, lo cual impulsará un aumento de la cantidad de dispositivos conectados. Esta tecnología potenciará la conexión entre objetos sin que sea necesaria la intervención de las personas a escala masiva. Esta nueva tecnología permitirá mayor velocidad, penetración, menor consumo de baterías y soporte de movilidad y voz, además de la estandarización, robustez, escalabilidad y seguridad en IoT.

6.1 Trabajo a futuro

El trabajo a futuro podría incluir, realizar la conexión de la herramienta actual con una red LoRa de manera que el sistema pusiese recibir directamente los datos provenientes de los sensores.

Utilizando Nginx, o alguna otra herramienta que permita colgar la herramienta de monitoreo en una dirección web, sería posible acceder al sistema de visualización a través de internet sin importar el lugar donde esté el servidor.

Una mejora a la herramienta actual sería implementar la autenticación mediante certificados en el MQTT Gateway, si bien, Mosquitto soporta este tipo de seguridad de forma nativa, el *gateway* de momento no es capaz de realizar la autenticación.

Bibliografía

Barton, R., Salgueiro, G., & Hanes, D. (2017). *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things* [Versión lector de libros electrónicos]. Recuperado de https://www.oreilly.com/library/view/iot-fundamentals-networking/9780134307091/

Docker Inc. (2019). *Docker container networking*. Recuperado de https://docs.docker.com/v17.09/engine/userguide/networking/

Docker Inc. (2019). *Dockerfile reference*. Recuperado de https://docs.docker.com/engine/reference/builder/

Docker Inc. (2019). *Overview of Docker Compose*. Recuperado de https://docs.docker.com/compose/overview/

Google Cloud. (2019, Junio 4). *Overview of Internet of Things*. Recuperado de https://cloud.google.com/solutions/iot-overview

Google Cloud. (2019, Marzo 11). *Remote monitoring and alerting for IoT*. Recuperado de https://cloud.google.com/solutions/remote-monitoring-and-alerting-for-iot

Hillar, G. (2017). *MQTT Essentials – A Lightweight IoT Protocol* [Versión lector de libros electrónicos]. Recuperado de https://learning.oreilly.com/library/view/mqtt-essentials-/9781787287815/

Iberdrola S.A. (2013). *What is the active energy, reactive energy and the power factor?*. Recuperado de http://help.customers.iberdrola.es/frequency-question/what-is-the-active-energy-reactive-energy-and-the-power-factor/

LoRa Server. (2019). *The LoRa Server project*. Recuperado de https://www.loraserver.io/overview/

Nginx. (2019). *What is NGINX?*. Recuperado de https://www.nginx.com/resources/glossary/nginx/

Ortiz, J. (2016). *Household Power Consumption*. Recuperado de https://data.world/databeats/household-power-consumption/activity

Prometheus. (2019). *Securing Prometheus API and UI Endpoints using basic auth*. Recuperado de https://prometheus.io/docs/guides/basic-auth/

Red Hat, Inc. (2019). *What is Docker?*. Recuperado de https://opensource.com/resources/what-docker

Revista ElectroIndustria. (2017, Marzo). La importancia de la medición y corrección del factor de potencia. Recuperado de http://www.emb.cl/electroindustria/articulo.mvc?xid=2976&srch=transformador&act=3

Rouse, M. (2019, Marzo). *Internet of Things (IoT)*. Recuperado de https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

Singh, H. (2018, Enero 31). *The Internet of Things: IoT Solutions and its Benefits*. Recuperado de http://customerthink.com/the-internet-of-things-iot-solutions-and-its-benefits/

Apéndice

Apéndice A. Contenedor MQTT-Mosquitto y MQTT Gateway

dockerfile

```
#Download base image ubuntu 16.04
FROM ubuntu:16.04

#Update Ubuntu Software repository
RUN apt-get update

#Install go1.12.1
RUN apt-get install sudo
RUN apt-get update && apt-get install -y \
curl
RUN apt-get update && apt-get install -y \
gcc make
RUN apt-get update && apt-get install -y \
wget
RUN apt-get update && apt-get install -y \
software-properties-common
RUN apt-get update && apt-get install -y \
git
RUN sudo add-apt-repository ppa:longsleep/golang-backports
RUN sudo apt-get update
RUN apt-get update && apt-get install -y \
golang-go

#Install Mosquitto packages
RUN apt-get update && apt-get install -y \
mosquitto
RUN apt-get update && apt-get install -y \
mosquitto-clients

#Install MQTTGateway
RUN go get -u github.com/inuits/mqttgateway

#Install Supervisor
RUN apt-get update && apt-get install -y \
supervisor
RUN mkdir -p /var/log/supervisor
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf

EXPOSE 1883 9337 9001

CMD ["/usr/bin/supervisord"]
```

supervisor.conf

```
[supervisord]
nodaemon=true


[program:mosquitto]
command=/usr/sbin/mosquitto
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true

[program:mqttgateway]
command=sudo root/go/bin/mqttgateway
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true
```

Apéndice B. Contenedor Prometheus y NGINX

dockerfile

```
#Download base image ubuntu 16.04
FROM ubuntu:16.04

#Update Ubuntu Software repository
RUN apt-get update
RUN apt-get install sudo
RUN apt-get update && apt-get install -y \
curl
RUN apt-get update && apt-get install -y \
software-properties-common

#Install Nginx
RUN \
  add-apt-repository -y ppa:nginx/stable && \
  apt-get update && \
  apt-get install -y nginx && \
  rm -rf /var/lib/apt/lists/* && \
  echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
  chown -R www-data:www-data /var/lib/nginx
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs",
"/etc/nginx/conf.d", "/var/log/nginx", "/var/www/html"]
WORKDIR /etc/nginx

#Install Supervisor
RUN apt-get update && apt-get install -y \
supervisor
RUN mkdir -p /var/log/supervisor
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
RUN cd


#Install Prometheus
RUN sudo useradd --no-create-home --shell /bin/false prometheus
RUN sudo mkdir /etc/prometheus
RUN sudo mkdir /var/lib/prometheus
RUN sudo chown prometheus:prometheus /etc/prometheus
RUN sudo chown prometheus:prometheus /var/lib/prometheus
RUN cd ~
RUN curl -LO
https://github.com/prometheus/prometheus/releases/download/v2.0.0/prometheu
s-2.0.0.linux-amd64.tar.gz
RUN tar xvf prometheus-2.0.0.linux-amd64.tar.gz
RUN sudo cp prometheus-2.0.0.linux-amd64/prometheus /usr/local/bin/
RUN sudo cp prometheus-2.0.0.linux-amd64/promtool /usr/local/bin/
RUN sudo chown prometheus:prometheus /usr/local/bin/prometheus
RUN sudo chown prometheus:prometheus /usr/local/bin/promtool
RUN sudo cp -r prometheus-2.0.0.linux-amd64/consoles /etc/prometheus
RUN sudo cp -r prometheus-2.0.0.linux-amd64/console_libraries
/etc/prometheus
RUN sudo chown -R prometheus:prometheus /etc/prometheus/consoles
RUN sudo chown -R prometheus:prometheus /etc/prometheus/console_libraries
```

```
RUN rm -rf prometheus-2.0.0.linux-amd64.tar.gz prometheus-2.0.0.linux-amd64
COPY prometheus.yml /etc/prometheus/
RUN sudo chown prometheus:prometheus /etc/prometheus/prometheus.yml


EXPOSE 80 443 12321
#EXPOSE 9090

CMD ["/usr/bin/supervisord"]
```

supervisor.conf

```
[supervisord]
nodaemon=true


[program:prometheus]
command = sudo -u prometheus /usr/local/bin/prometheus --config.file
/etc/prometheus/prometheus.yml --storage.tsdb.path /var/lib/prometheus/ --
web.console.templates=/etc/prometheus/consoles --
web.console.libraries=/etc/prometheus/console_libraries --web.external-
url=http://172.19.0.1:443/
#command = sudo root/go/bin/prometheus --web.route-prefix="/"
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true

[program:nginx]
command=nginx
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true
```

nginx.conf

```
http {
    server {
        listen 443;

        location / {
            auth_basic           "Prometheus";
            auth_basic_user_file /etc/nginx/.htpasswd;

            proxy_pass           http://172.19.0.1:9090/;
        }
    }
}

events {
}
```

htpasswd

```
Prometheus:$apr1$Ejlg8Zn4$7tGot7a8AzYIpTJ6Umsq9/
```

prometheus.yml

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds.
Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default
is every 1 minute.
  # scrape_timeout is set to the global default (10s).

  # Attach these labels to any time series or alerts when communicating
with
  # external systems (federation, remote storage, Alertmanager).
  #external_labels:
   #   monitor: 'codelab-monitor'

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
#rule_files:
  # - "first.rules"
  # - "second.rules"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries
scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['172.19.0.1:9090']

  - job_name: 'mqttgateway'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['172.19.0.1:9337']
```

Apéndice C. Fichero para puesta en marcha del sistema

docker-compose.yml

```
version: '2'

services:
  mosquitto-mqttgateway:
    container_name: mosquitto-mqttgateway
    image: test
    ports:
      - "1883:1883"
      - "9337:9337"
    networks:
      mynet:
        ipv4_address: 172.19.0.8

  grafana:
    container_name: grafana
    image: grafana/grafana
    ports:
      - "3000:3000"
    depends_on:
      - prometheus-nginx
    networks:
      mynet:
        ipv4_address: 172.19.0.10

  prometheus-nginx:
    container_name: prometheus-nginx
    image: prometheus_nginx
    container_name: prometheus-nginx
    volumes:
      - "/Users/prometheus.yml:/etc/prometheus/prometheus.yml"
      - "/Users/nginx.conf:/etc/nginx/nginx.conf"
      - "/Users/htpasswd:/etc/nginx/.htpasswd"
    ports:
      - "12321:12321"
      - "9090:9090"
      - "80:80"
      - "443:443"
    networks:
      mynet:
        ipv4_address: 172.19.0.12

networks:
  mynet:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.19.0.0/24
          gateway: 172.19.0.1
```

Apéndice D. Programa para simular sensor

pseudo_sensor_IoT.yml

```
import csv
import datetime
import paho.mqtt.client as mqtt #in order to use the client class
import sys
import os.path
import time


def main():
    client = mqtt.Client(client_id='client01', clean_session=False)
    client.connect(host='127.0.0.1', port=1883)

    csv.register_dialect('myDialect', delimiter = ',',
quoting=csv.QUOTE_ALL, skipinitialspace=True)

    with open('household_power_consumption_edited.csv', 'r') as csvFile:
        reader = csv.reader(csvFile, dialect='myDialect')
        next(reader)
        for row in reader:
            time.sleep(2)
            value = client.publish("prometheus/voltage", row[4])
            value = client.publish("prometheus/globalreactivepower",
row[3])
            value = client.publish("prometheus/globalactivepower", row[2])

        csvFile.close()

    with open('beach-weather-stations-automated-sensors-1_edited.csv', 'r')
as csvFile:
        reader = csv.reader(csvFile, dialect='myDialect')
        next(reader)
        for row in reader:
            # datetime_value = time_tango(row[0], row[1])
            time.sleep(2)
            if row[0] == "Oak Street Weather Station":
                value = client.publish("prometheus/oakst_airtemperature",
row[2])
                value = client.publish("prometheus/oakst_humidity", row[4])
                value = client.publish("prometheus/oakst_totalrain",
row[7])
                value =
client.publish("prometheus/oakst_precipitationtype", row[8])
                value = client.publish("prometheus/oakst_batterylife",
row[15])

            if row[0] == "Foster Weather Station":
                value = client.publish("prometheus/foster_airtemperature",
row[2])
                value = client.publish("prometheus/foster_humidity",
row[4])
                value = client.publish("prometheus/foster_totalrain",
row[7])
                value =
client.publish("prometheus/foster_precipitationtype", row[8])
```

```
                value = client.publish("prometheus/foster_batterylife",
row[15])


        csvFile.close()


if __name__ == '__main__':
    main()

sys.exit(0)
```

Apéndice E. Fichero *bash* para arrancar el sistema

tfm_deploy

```
echo "----- Run containers -----------------------"
docker-compose up -d --no-recreate
echo

echo "----- Run Python script --------------------"
python pseudo_sensor_IoT.py
echo

echo "----- Open Grafana -------------------------"
open http://localhost:3000/
echo
```